ピクチャ・パターン照合アルゴリズム(計算アルゴリズムと計算量の基礎理論)

AUTHOR(S):

竹田, 正幸

CITATION:

竹田, 正幸. ピクチャ・パターン照合アルゴリズム(計算アルゴリズムと計算量の基礎理論). 数理解析研究所講究録 1989, 695: 233-242

ISSUE DATE:

1989-06

URL:

http://hdl.handle.net/2433/101385

数理解析研究所講究録
第695巻 1989年 233-242

ピクチャ・パターン照合アルゴリズム

竹田 正幸 (Masayuki Takeda)

九州大学大学院総合理工学研究科
情報システム学専攻

要旨

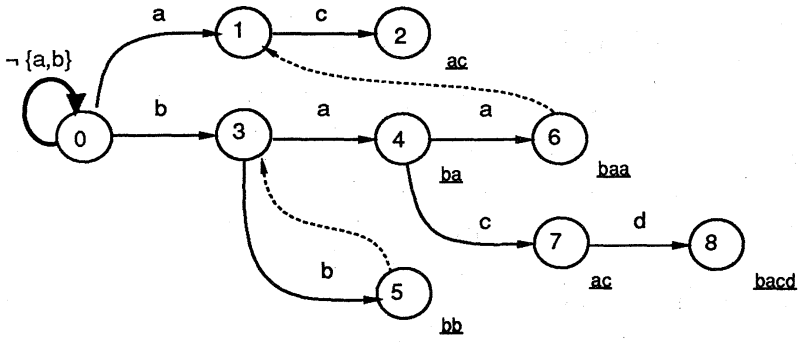
本稿では，従来の文字列パターンだけでなく，文字とピクチャの混在するパターンを対象としたパターン照合問題を取り扱う．たとえば，英小文字(a,b,...,z)を表すピクチャをA，数字(0,1,...,9)を表すピクチャをNとするとき，ab$A$, a7$NNNNA$などのパターンをテキスト中より検出することを考える．著者は既に，こうしたピクチャを含むパターンを複数個扱うための高速な照合アルゴリズムを提案している．このアルゴリズムは，文字列パターンに対する効率的な照合アルゴリズムの一つとして知られるAho-Corasick法（AC法）を自然に拡張したものであり，AC法と同様，与えられたパターンの集合からパターン照合機械とよばれる一種の有限オートマトンを構成し，テキスト上でそれを走らせることにより，同時に複数個のパターンの照合が可能である．また，AC法の別の変形として，Arikawa-Shiraishiによって，パターン照合機械を用いた複数文字列同時置き換え用のアルゴリズムが考案されている．このアルゴリズムは，テキスト上を左から右へただ1度走査する間に複数組の置き換えを同時に行う，非常に効率的なものである．そこで本稿では，この二つのアルゴリズムを組み合わせることにより，ピクチャを含むパターンによる複数パターンの同時置き換えアルゴリズムを提案する．

1 はじめに

従来のパターン照合問題で対象とするパターンは文字列であった．本稿では，文字列パターンだけでなく，文字とピクチャの混在するパターンを対象としたパターン照合問題を取り扱う．たとえば，英小文字(a,b,...,z)を表すピクチャをA，数字(0,1,...,9)を表すピクチャをNとするとき，ab$A$, a7$NNNNA$などのパターンをテキスト中より検出することを考える．ここでピクチャは文字の種類の限定されたワイルド・カードとみなせるから，こうした問題はある種の曖昧さを許したパターンの照合として捉えることができよう．

文字列パターンを複数個同時に扱う効率的なアルゴリズムとしてAho-Corasick法（AC法）[1]が知られている．AC法では，まず与えられた複数のパターンからパターン照合機械とよばれる一種の有限オートマトンを構成する（図1参照）．これにより，テキストをただ一度走査する間に同時に複数のパターンを照合することができる．なお，AC法については文献[1] [3]を参照されたい．

著者は既に，このAC法を拡張することにより，ピクチャを含むパターンを同時に複数個扱う高速なパターン照合アルゴリズムを提案している[4] [5]．このアルゴリズムは，文字列パターンを対象とし



実線矢印は goto 関数，破線矢印は failure 関数を，それぞれ表す．ただし状態 2,3,4,7,8 から状態 0 への破線は省いてある．また，各状態の右下の下線付文字列はその状態からの output を表す．

図 1: Aho-Corasick 型パターン照合機械

た場合には AC 法によるものと同じ照合機械を構成することができる．また，テキスト走査アルゴリズムは AC 法とまったく同じになる．こうした意味で，このアルゴリズムは AC 法の自然な拡張になっている．

ところで，テキストを編集する際には，ある文字列を対応する別の文字列で置き換える必要がたびたび生じる．Arikawa-Shiraishi [2] は，複数個の対の置き換えを，テキストをただ 1 回走査する間に同時に行う効率的なアルゴリズムを考案している．このアルゴリズムは通常の AC 法を変形したものである．

そこで本稿では，これら二つのアルゴリズムを組み合わせることにより，ピクチャを含むパターンによる複数パターンの同時置き換えアルゴリズムを提案する．たとえば，1967, 1979, 1985, ... をそれぞれ '67, '79, '85, ... へ置き換えたい場合を考えよう．上述の複数文字列置き換えアルゴリズムを用いると，置き換えのすべての対 (1967, '67), (1979, '79), (1985, '85), ... を与えなければならず，照合機械の状態数もそれに応じて増大する．これに対し，ピクチャの考え方を使えば，

$$19NN \Rightarrow 'NN$$

のような置き換えの形式を与えるだけでよく，状態数も少なくてすむ．

2 ピクチャ・パターン照合問題

まず，パターン照合問題を次のように拡張する．

定義 1 ピクチャの族を$\Delta = \{A_1, A_2, \ldots, A_p\}$とする．ただし，$A_i \subseteq \Sigma$, $A_i \neq \emptyset$, $A_i \cap A_j = \emptyset$ $(i \neq j)$である．パターンは$(\Sigma \cup \Delta)^+$の要素の中から選ぶものとする．以上の準備のもとで，（拡張された）パターン照合問題を次のように定義する：

> パターン$X_1X_2 \ldots X_m$ （$X_i \in \Sigma \cup \Delta$）とテキスト$a_1a_2 \ldots a_n$ （$a_i \in \Sigma$）が与えられたとき，$a_ja_{j+1} \ldots a_{j+m-1} \in X_1X_2 \ldots X_m$となるような$j$をすべて求めよ．

上の定義で，特にパターンをΣ^+の要素に限定すれば通常のパターン照合問題になる．なお，簡単のため今後も$\{a\}$とaを同一視することがある．

さて，この問題をAho-Corasick型のパターン照合機械を用いて解くことを考えよう．パターンがピクチャのみからなる場合は非常に単純である．実際，各ピクチャを1個の記号とみなしてパターン照合機械を構成すればよい．しかし，パターン中に文字とピクチャが混在する場合や，あるいは文字列パターンと同時に探索したい場合には，いろいろと困難な問題が生じてくる．以下の例では，

$$A = \{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}\}, \quad N = \{0, 1, \ldots, 9\}, \quad \Sigma = A \cup N, \quad \Delta = \{A, N\}$$

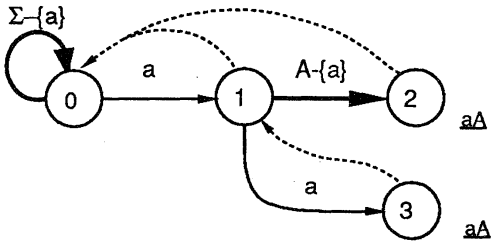
とする．

まず，ピクチャを文字に分解して，パターンに属する文字列すべてをキーワードとみなしてパターン照合機械を構成する方法が考えられる．つまり，パターンが$\mathtt{a}A$であれば，$\mathtt{aa}, \mathtt{ab}, \ldots, \mathtt{az}$をキーワードとして照合機械を構成するわけである．しかし，たとえばパターンA^mに対しては状態数は

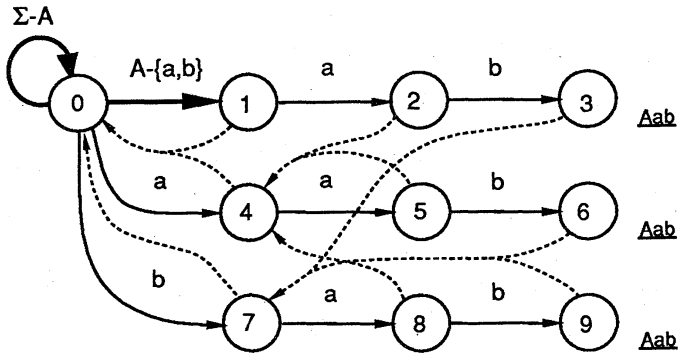
$$1 + 26 + 26^2 + \ldots + 26^m = \frac{26^{m+1} - 1}{25}$$

となる．この方法はこのように状態数の爆発を引き起こす．

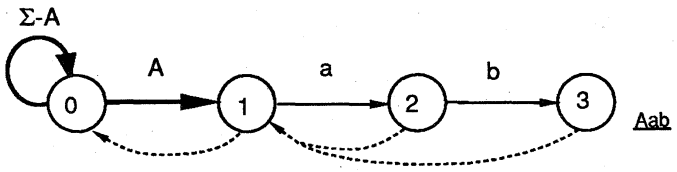
次に考えられる方法としては，まず初めにピクチャの族とパターン中の固定文字から互いに重ならない部分ピクチャの族を決定しておいて，それをもとにパターン照合機械を構成する方法がある．この方法によると，パターン$\mathtt{a}A$に対しては，ピクチャAを$\{\mathtt{a}\}$と$A - \{\mathtt{a}\}$の二つに分割するので，次のような照合機械が構成される．



しかし，たとえばパターンが$A\mathtt{ab}$である場合を考えると，ピクチャAを$\{\mathtt{a}\}$, $\{\mathtt{b}\}$, $A - \{\mathtt{a},\mathtt{b}\}$の三つに分割するので，構成される照合機械は，



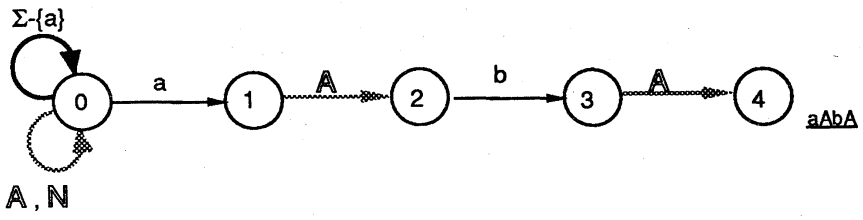
のようになる．この照合機械は冗長である．実際，パターン Aab に対しては，



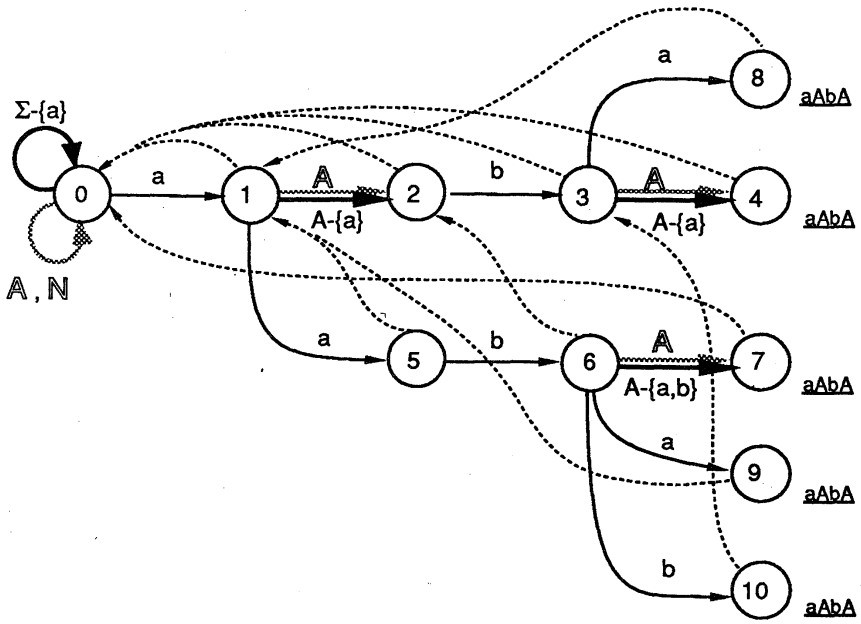
のような照合機械で十分である．つまりこの場合には，状態 0 からの goto 遷移に関してピクチャ A の各文字を区別する必要がないため，goto 分岐を行わなくてよいわけである．

そこで，1 個のピクチャに属する各文字に対して，区別が必要な場合だけ別の状態へ遷移するように goto 関数を定めることを考える．それには failure 関数を構成する際に，failure 遷移の行先を別にする必要のある場合にのみ goto 遷移を分岐させればよい．

著者の考案したアルゴリズム [4] [5] は，このようなアイデアに基づくものである．このアルゴリズムは，たとえばパターンを aAbA とすると，まず，ピクチャを 1 個の記号とみなし（アウトライン文字で表わす），



のような goto 遷移グラフを構成する．そして幅優先順に節点を探索しながら failure 関数を構成していき，必要に応じて goto 遷移を枝分かれさせる．すると



のような照合機械を構成することができる．パターンが複数個ある場合にも同様の考えを用いることにより，このような冗長性の少ないパターン照合機械を構成することができる．また，failure関数を除去して決定性の有限オートマトンに変形することもAC法と同様きわめて容易である．

なお，このアルゴリズムの詳細や，その妥当性，および計算量については[4] [5]を参照されたい．

3 複数文字列同時置き換え

本節では，Arikawa-Shiraishi [2] による複数文字列同時置き換えアルゴリズムについて簡単に紹介する．

テキスト処理において，ある文字列を対応する他の文字列で置き換える必要が頻繁におこる．通常，こういった仕事は編集用コマンド

CHANGE /x/y/ ALL

やそれに類するコマンドを繰り返し用いることによって実行される．上のコマンドCHANGEは，テキスト中に現れるすべての文字列xを文字列yで置き換えることを意味している．しかし，置き換えるべき文字列の対(x,y)がn個あるときにはこのコマンドを少なくともn回用いなければならない．したがって，計算機自身もn回テキストを走査することになる．Arikawa-Shiraishiのアルゴリズムは，このようなn個の対の置き換えを，テキストを1回走査する間に繰り返し行うものである．

こうした複数文字列同時置き換え用のパターン照合には，パターンの重なり具合に関して問題が生じる．すなわち，テキスト中でパターンが重なりあって出現している場合に，どのパターンを優先して

置き換えるべきか，という問題である．このアルゴリズムは，次のようなルールに基づいた置き換えを行う．

(1) 重なりあったパターンのうち，照合点（パターンの始まっている位置）の若いものを優先して置き換える．

(2) もし，同じ照合点をもつ複数のパターンが出現していれば，そのうちの最長のものを置き換える．

ここで用いられる複数文字列同時置き換え用のパターン照合機械（以後，rpmm と略記する）は，AC 型パターン照合機械を変形したものである．rpmm は goto 関数 g，failure 関数 f，output 関数 out からなる．このうち goto 関数と failure 関数は AC 法によるものとほぼ同じ役割をするが，output 関数は置き換えたテキストを出力するためのものである．アルゴリズムは rpmm を構成するための部分とそれを走らせて置き換えを行う部分とに分かれる．

rpmm を構成する部分では，たとえば，入力 $\Pi = \{$ (ABCDE, α), (CDE, β), (BC, γ) $\}$ に対しては，図 2のような rpmm を構成する．図において，破線は falure 関数による遷移を表す．ただし，状態 0, 2, 3 を除く状態から状態 0 へ至る failure 遷移はすべて省略している．

こうした rpmm を用い，次のように置き換えを行う．rpmm は failure 遷移を行うごとに output 関数を用いて出力文字列を出す．ただし，状態 0 から状態 0 へ goto 遷移を行う際にはそのときの入力文字を出力する．また，テキストを読み終えたときの状態が 0 でなければ，状態 0 にもどるための failure 遷移を繰り返し，そのつど出力文字列を出す．図 2の rpmm はテキスト “DEABCCBCE” 上で，図 3のように動作する．

rpmm の構成時間，走査時間はもとの AC 法と同様に，いずれも線形時間である．なお，このアルゴリズムの詳細については，文献 [2] を参照されたい．

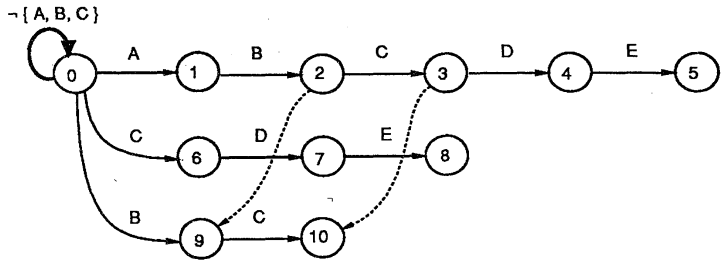
4 文字列パターンの場合の妥当性

まず，前節で紹介した文字列パターンの置き換えアルゴリズム [2] の妥当性について論じておかなければならない．というのは，論文 [2] の与えている妥当性の証明は，failure 関数，output 関数の特徴づけに関してまったく誤っているからである．このアルゴリズムをピクチャを含むパターンへ拡張するためには，この二つの関数を正しく特徴づけておくことが不可欠である．

Aho-Corasick [1] は，AC 法で用いる failure 関数を特徴づける補題を与えている．しかし，前節で示した rpmm の failure 関数は，その補題で特徴づけることはできない．二つのアルゴリズムによる failure 関数の構成法の違いを見てみよう．AC 法の failure 関数 f は，次のようにして再帰的に構成される．

初期段階 深さ 1 の状態 s に対して $f(s) := 0$ とする．

再帰段階 深さ 2 以上の状態 s に対し，その親 r を考える．いま，$g(r, c) = s$ であったとする．状態 r から failure 遷移を繰り返して到達できる状態 fst で文字 c による goto パスをもつ（つまり，



(a) goto 関数と failure 関数

st	*out*(*st*)	
0	無定義	
1	A	
2	A	(*out*(1))
3	A	
4	AαD	(*out*(3) · *out*(10)·D)
5	α	
6	C	
7	CD	(*out*(6)· D)
8	β	
9	B	
10	γ	

(b) output 関数

図 2: rpmm の完成

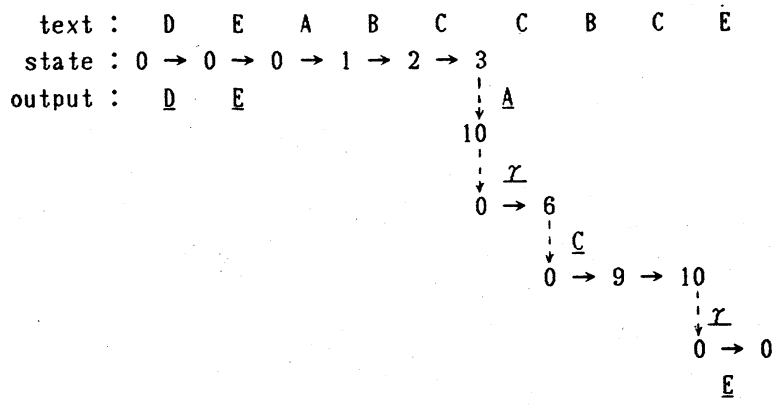


図 3: rpmm の動作例

$g(fst, c) \neq$ fail である）もののうち，状態 r に最も近い fst を選び，$f(s) := g(fst, c)$ と定める．

これに対して，rpmm の failure 関数の構成では，初期段階でパターンの右端に対応する状態 s に対しても $f(s) := 0$ としている．再帰段階は，すでに failure 関数の定義されている状態を除き，AC 法と同様である．

この初期段階における両者の違いは，そこから構成される failure 関数の性質にどのような違いをもたらすだろうか．著者はこの failure 関数の特徴づけ補題を与え，さらに rpmm の妥当性の正しい証明を与えた．しかしここでは紙数の都合により省略せざるを得ない．詳細については文献 [6] を参照されたい．

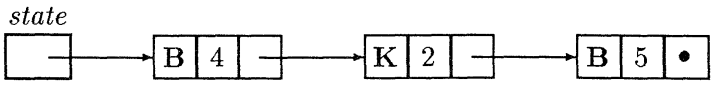
5 ピクチャを含む場合の問題点

キーパターンにピクチャを含むことを許す場合，いくつかの問題が生じる．この節ではそれらについて論じる．なお今後は，パターン α の出現を β で置き換える場合，α を**キーパターン**あるいは単にパターンとよび，β を**出力パターン**とよぶことにする．

第 1 の問題は，置き換えるべきキーパターンの優先順序に関するものである．たとえば，キーパターンとして aA，Aa があるような場合文字列 aa はこのどちらにもヒットする．このとき，文字列 aa はどちらのパターンの出現とみなして置き換えるべきであろうか．このことを決めるための自然な基準はないように思われる．そこで，こうした場合にはユーザーが優先順を与えることにする．最も単純には，ユーザーが置き換えの対を登録して行く際に後（あるいは先）に与えた対を優先させることが考えられる．

第 2 の問題は，パターンがピクチャを含む場合には，pmm の状態 st が与えられても，状態 0 からそこに到達するまでに読んだ（st の深さ分の）文字列を復元することができないということである．この問題は最低 pmm の深さ分のバッファを用意することで解決できる．しかし，テキスト走査時におい

てfailure遷移の度に出力する文字列の一部をバッファから読み出すように変更しなければならない．したがって，output関数は単なる文字列ではなく，たとえば次のようなリスト構造をもつものになる．



図で，Bはバッファの意であり，その隣の数字は文字数を表わす．また，Kはキーパターンの意であり，その隣の数字はその番号を表わす．よって，このリスト全体は，バッファから4文字分出力して，次に2番のキーパターンに対する置き換えを行い（この間バッファのポインタをキーパターン長だけ進めておく），再びバッファから5文字分出力する，ということを表わしている．output関数をこのように構成すれば，キーパターンがピクチャを含む場合のrpmmを構成することができる．

第3の問題は，キーパターンの出現と置き換えて出力する出力パターンに関するものである．この出力パターンが固定文字列であれば問題はないが，たとえば，1節で挙げた置き換えの例

$$19NN \Rightarrow \text{'}NN$$

において，ピクチャは一種の変数として働く．つまり厳密には

$$19N_1N_2 \Rightarrow \text{'}N_1N_2$$

という置き換えを指定したものと解釈されている．このようにピクチャを変数と考えてしまえば，

$$A_1A_2\mathtt{ab}A_3 \Rightarrow A_3\mathtt{c}A_1\mathtt{d}A_2$$

のような置き換えも可能になる．

ここではアルゴリズムの詳細にふれる余裕はないが，キーパターン$19NN$に対する照合機械の例を図4に示しておく．ただしoutput関数は省いてある．

さて，ここで

$$19NN \quad \Rightarrow \quad (19NN)$$

という置き換えを考えてみよう．たとえば，参考文献中などで論文の出版年を括弧でくくりたい，というような場合である．もしテキスト中に，既に括弧でくくられているものが混在していたとすると，上記の置き換えを単純に行っただけでは“((1987))”のように二重に括弧の付いたものができてしまうことになる．しかし，本アルゴリズムはArikawa-Shiraishiのアルゴリズムの拡張になっているので，

$$\begin{aligned} 19NN \quad &\Rightarrow \quad (19NN)\,, \\ (19NN) \quad &\Rightarrow \quad (19NN) \end{aligned}$$

という2組の置き換えを指定すれば，意図した通りの置き換えを行うことができる．

なお，アルゴリズムの詳細およびその妥当性については文献[6]を参照されたい．

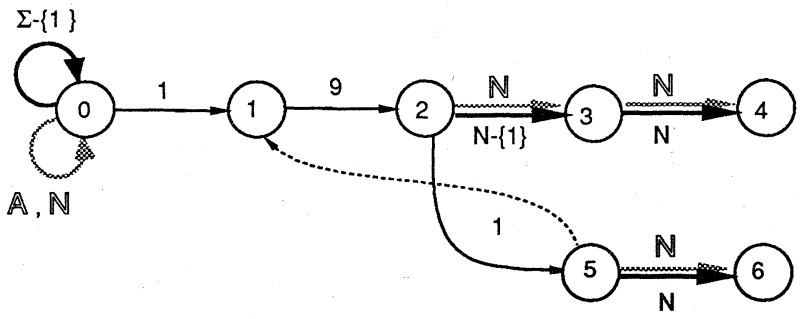


図 4: 19NN に対する照合機械

6 おわりに

ピクチャを含む複数のパターンを対象としたパターン照合問題を考え，そのための効率的なアルゴリズム [4] をもとにして，複数パターンの置き換えの指定にピクチャを含むことを許したアルゴリズムを考案した．これによって，文書の作成・編集に際し，より細やかな置き換えを行うことができよう．文字列パターンの置き換えとは異なり，細かな仕様をどうするかという問題が残っているため，まだ実現の段階には至っていないが，これは今後の課題である．

参考文献

[1] Aho, A. V. , Corasick, M. J. : Efficient String Matching : An Aid to Bibliographic Search, *Comm. ACM*, Vol. **18**, No. **6** (1975), pp. 333-340.

[2] Arikawa, S. , Shiraishi, S. : Pattern Matching Machines for Replacing Several Character Strings, *Bull. Inform. Cybern.* , Vol. **21**, No. **1-2** (1984), pp. 101-111.

[3] 有川節夫, 篠原武 : 文字列パターン照合アルゴリズム, コンピュータソフトウェア, Vol. 4, No. **2** (1987), pp. 2-23.

[4] 竹田正幸 : 固定文字と文字種の混在するパターンを対象とした Aho-Corasick 型パターン照合機械の構成法, 九州大学大型計算機センター 計算機科学研究報告, No. **6** (1989).

[5] Takeda, M. : A Fast Matching Algorithm for Pattens with Pictures, *Tech. Rep.* RIFIS-TR-CS-11, Reserch Institute of Fundamental Information Science, Kyushu Univ. (1989).

[6] Takeda, M. : Efficient Multiple String Replacing with Pictures, *Tech. Rep.* RIFIS-TR-CS-13, Reserch Institute of Fundamental Information Science, Kyushu Univ. (1989).